Kinetic Heaps

K, Tarjan, and Tsioutsiouliklis

Outline

Broadcast Scheduling

Parametric and Kinetic Heaps

Broadcast Scheduling Using a Kinetic Heap

The Computational Geometry View

What is known

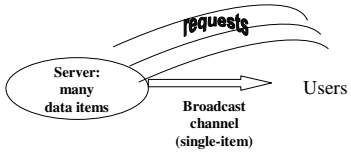
Four New Structures

- Simple Heap
- Balanced Heap
- Fibonacci Heap
- Incremental Heap

Open problems

Broadcast Scheduling

(Lecture by Mike Franklin ——— research papers)



One server, many possible items to send (say, all the same length)

One broadcast channel.

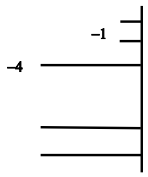
Users submit requests for items.

Goal: Satisfy users as well as possible, making decisions on-line. (say, minimize sum of waiting times)

Scheduling policies

Greedy = Longest Wait first (LWF):
Send item with largest sum of waiting times.

R x W: send item with largest (# requests x longest waiting time)



(vs. number of requests or longest single waiting time)

Questions (for an algorithm guy or gal)

LWF does well compared to what?

⇨ Try a competitive analysis

Open question 1

Can we improve the cost of LWF?

⇨ What data structure?

Will talk about this

Parametic Heap

A collection of **items**, each with an associated **key.**

key (i) = $a_i x + b_i$ a_i, b_i reals, x a real-valued parameter
a_i = slope, b_i = constant

Operations:

make an empty heap.

insert item i with key $a_i x + b_i$ into the heap: *insert*(i,a_i,b_i)

find an item i of minimum key for x = x_0: *find-min*(x_0)

delete item i : *delete*(i)

Kinetic Heap

A parametric heap such that successive x-values of *find mins* are non-decreasing.

(Think of x as time.)

x_c = largest x in a *find min* so far (current time)

Additional operation:

decrease the key of an item i, replacing it by a key that is no larger for all $x \geq x_c$: *decrease-key*(i,a,b)

Broadcast scheduling via kinetic heap

Need a max-heap (replace *find min* by *find max*, *decrease key* by *increase key*, etc)

Can implement LWF or R x W or any similar policy:

Broadcast decision is *find max* plus *delete*

Request is *insert* (if first) or *increase key* (if not)

Only *find max* need be real-time, other ops can proceed concurrently with broadcasting

Slopes are integers that count requests

Broadcast scheduling via kinetic heap (Cont.)

LWF:

Suppose a request for item i arrives at time t_s

If i is inactive then *insert*(i, $t-t_s$)
If i is active with key $at+b$ then *increase-key*(i, $(a+1)t+(b-t_s)$)

To broadcast an item at time t_s we perform *delete-max*(t_s)
and broadcast the item returned.

What is known about parametric and kinetic heaps?

A **key** is a **line** ⇨ computational geometry

Equivalent problems:

maintain the lower envelope of a collection of lines in 2D

| projective duality

maintain the convex hull of a set of points in 2D under insertion and deletion

"kinetic" restriction = "sweep line" query constraint

Results I

Overmars and Van Leeuwen (1981)
O(log n) time per query
$O(\log^2 n)$ time per update, worst-case

Chazelle (1985), Hershberger and Suri (1992)
(deletions only)
O(log n) time per query, worst-case
O(n log n) for n deletions

Results II

Chan (1999)
Dynamic hulls and envelopes
O(log n) time per query
$O(\log^{1+\varepsilon} n)$ time per update, amortized

Brodal and Jacob (2000)
O(log n) time per query
O(log n log log log n) time per insert,
O(log n log log n) timer per delete, amortized

Results III

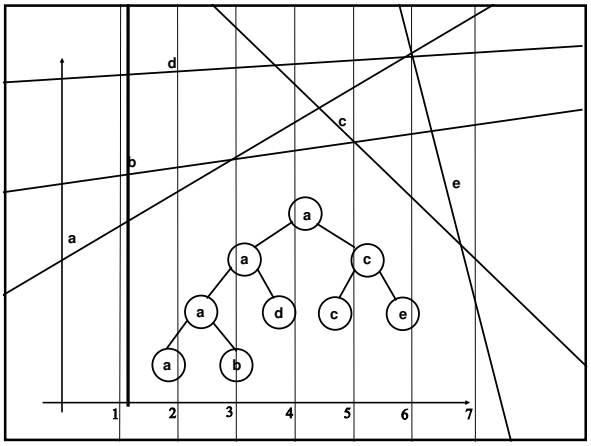
Basch, Guibas, and Hershberger (1997)

"Kinetic" data structure paradigm

Simple heap (details)

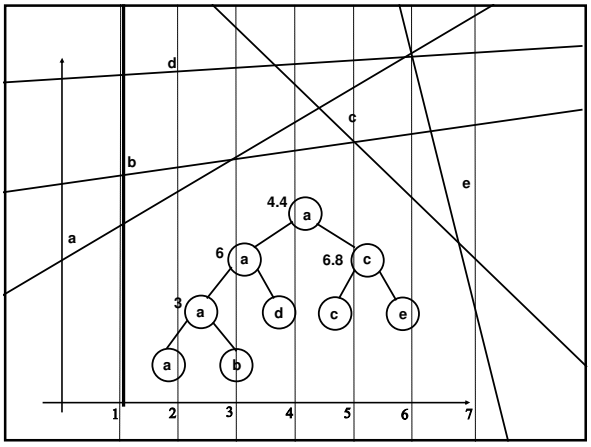
A balanced binary tree, with items in the leaves in left-right order by decreasing key slope.

The tree is a tournament on items by current key.



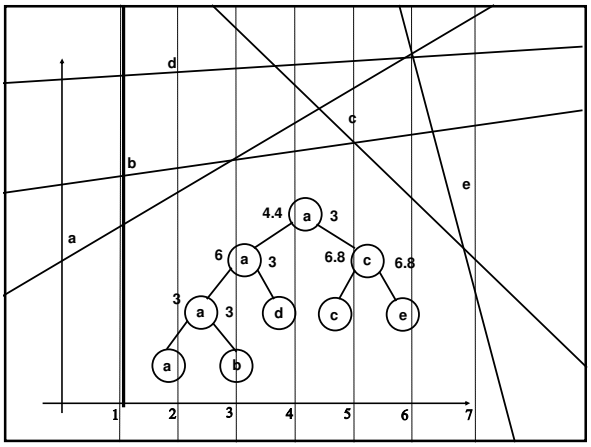
Simple kinetic heaps (Cont)

Add swap times at internal nodes



Simple kinetic heaps (Cont)

Add minimum future swap time



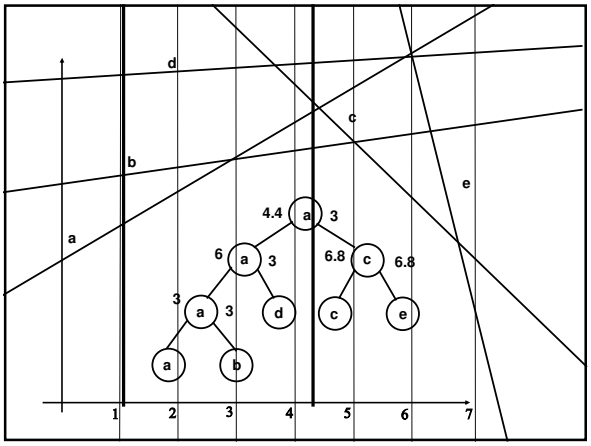
Simple kinetic heaps (Cont)

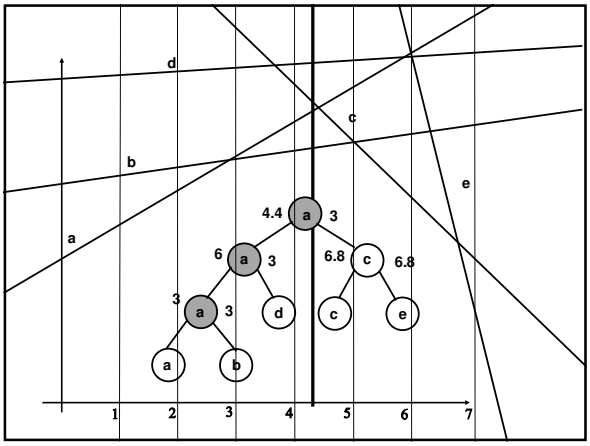
When we do a find-min if the current time moves forward, we act as follows:

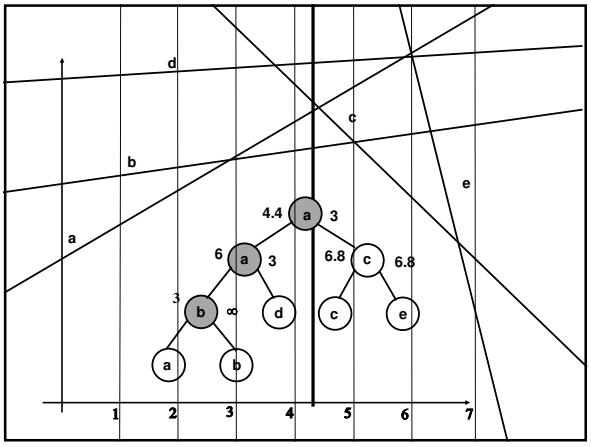
Locate all nodes whose winner changes.

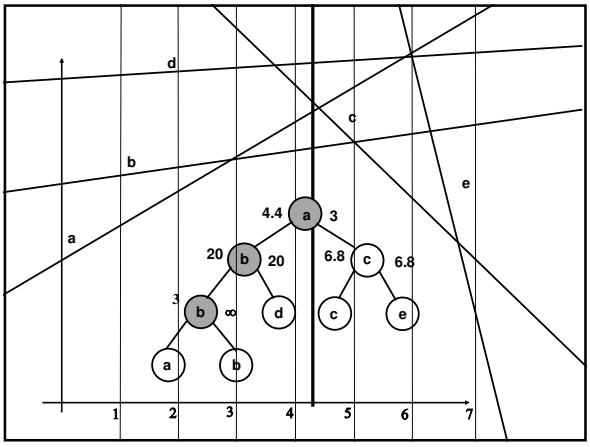
The paths from the root to those nodes form a subtree.

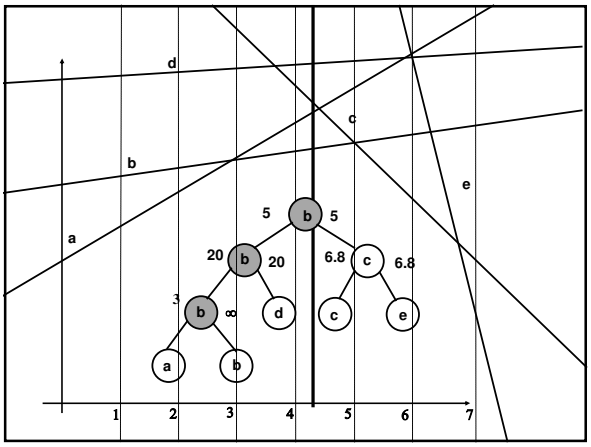
Update this subtree

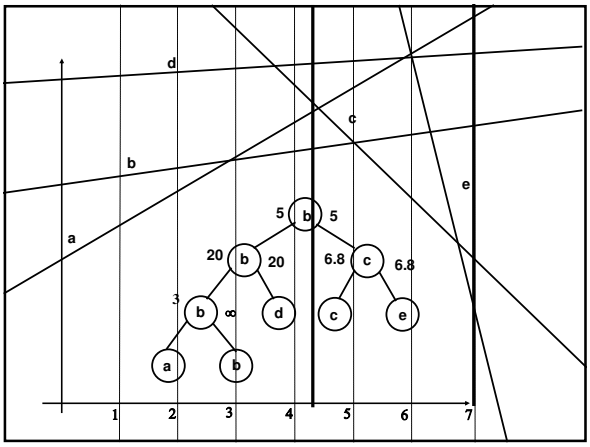


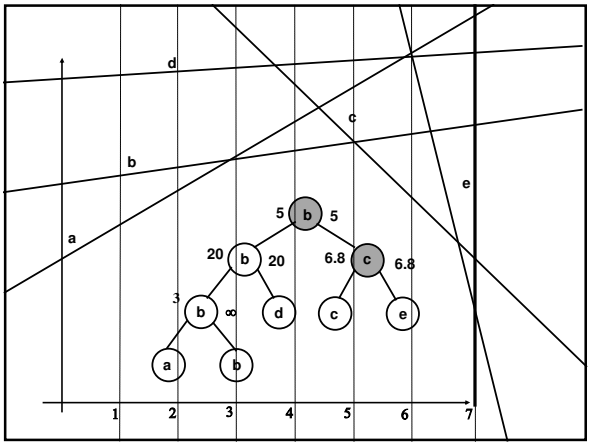


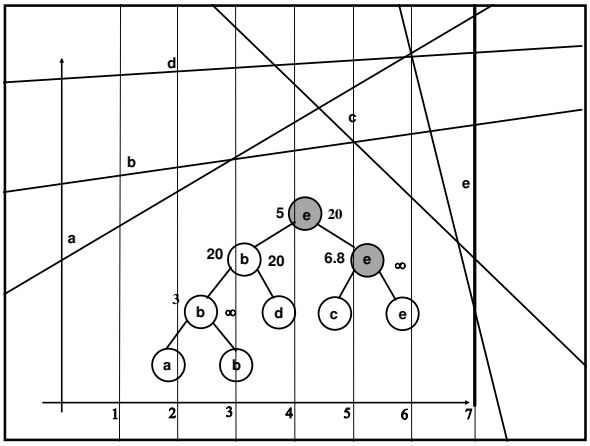


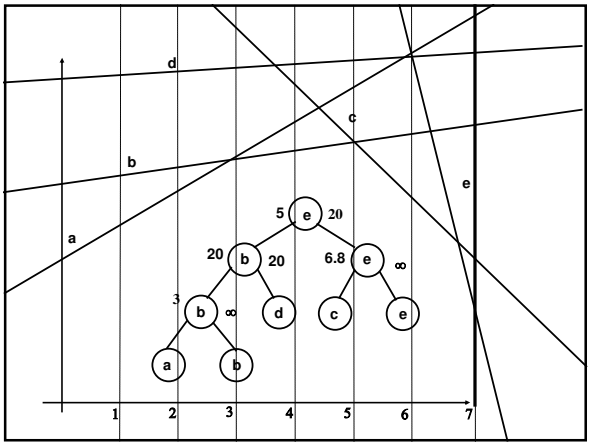






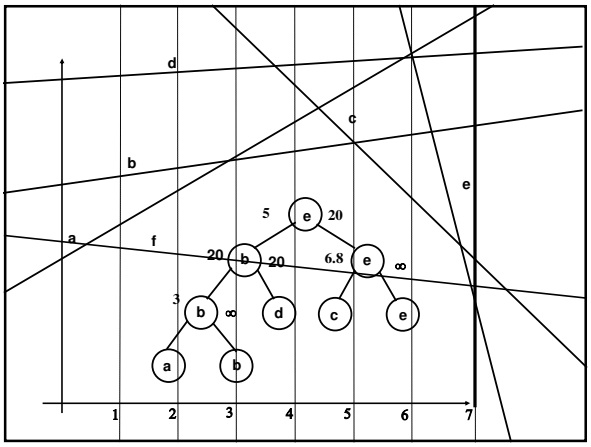


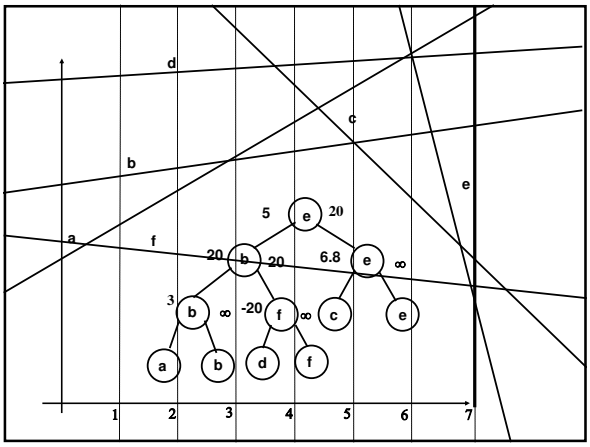


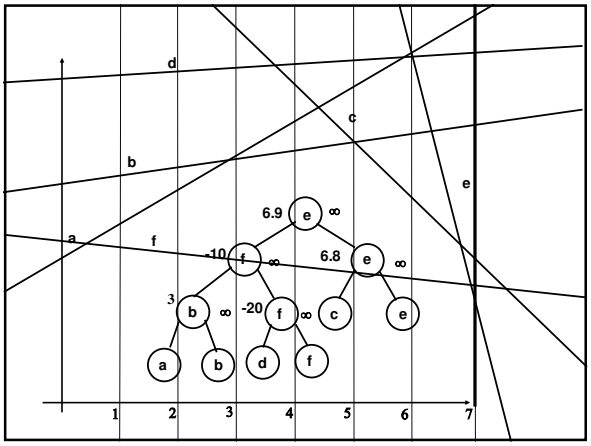


Simple kinetic heaps – insert & delete

Use the regular mechanism of (say) red-black trees







Simple kinetic heaps (analysis)

Find-min takes O(1) time if we do not advance the current time.

Advancing the current time can be expensive on the worst-case

Amortization:

Assign O(log n) potential to each node whose swap time is in the future.

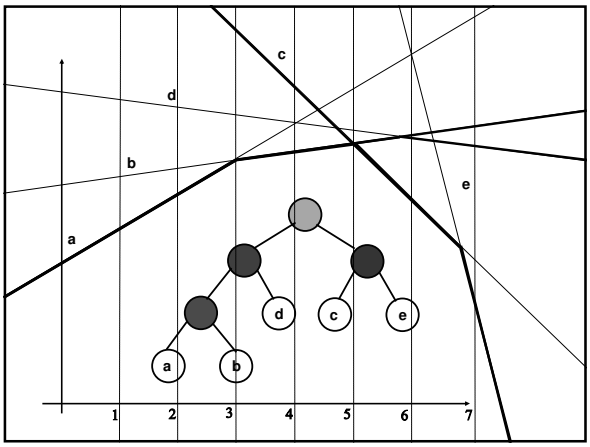
This potential pays for advancing the time.

So find-min takes O(1) amortized time.

Insert and delete take $O(\log^2 n)$ amortized time

How do we make this a parametric heap ?

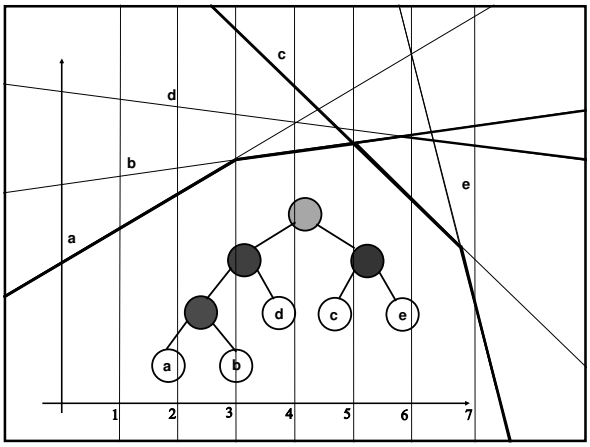
(Overmars-Van Leeuwen)



Looks like we use non linear space ?

Each node keeps the chain of segments that are on its envelope but not on its parent envelope.

This makes the space linear.



Analysis

Using search trees to represent the envelopes sorted by slops then:

Find-min takes O(log n) time

Insert and delete take $O(\log^2 n)$ time: You split and concatenate O(log n) such envelopes.

Linear space

Balanced kinetic heaps

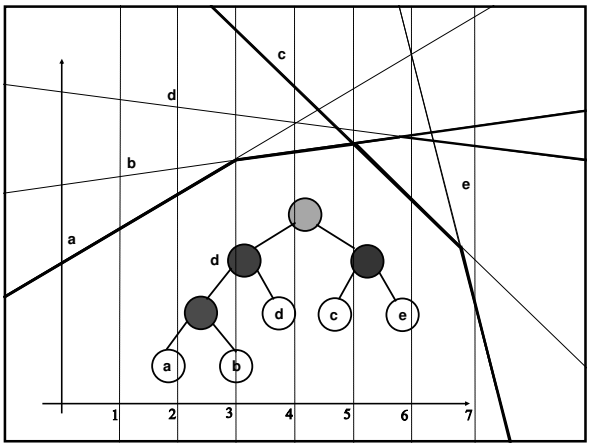
Plan: first we'll do only deletions faster and then we'll add insertions.

Our starting point is the previous data structure.

Def: The level of a line is the depth of the highest node where it is stored.

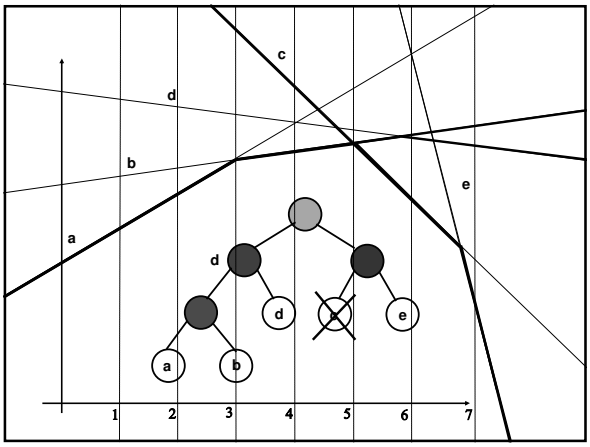
Deletion-only data structure

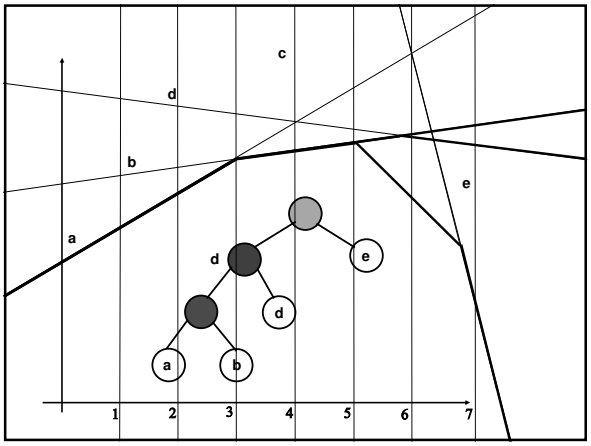
The level of a line is the depth of the highest node where it is stored.

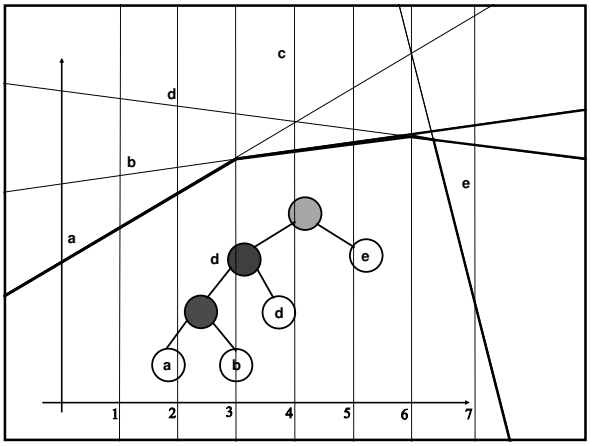


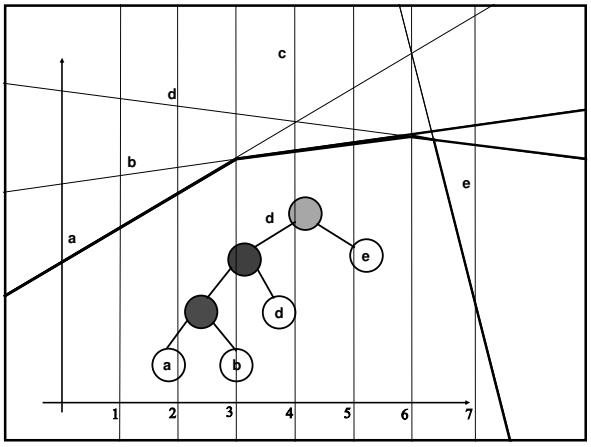
Deletion-only data structure

What happens when we delete a line ?



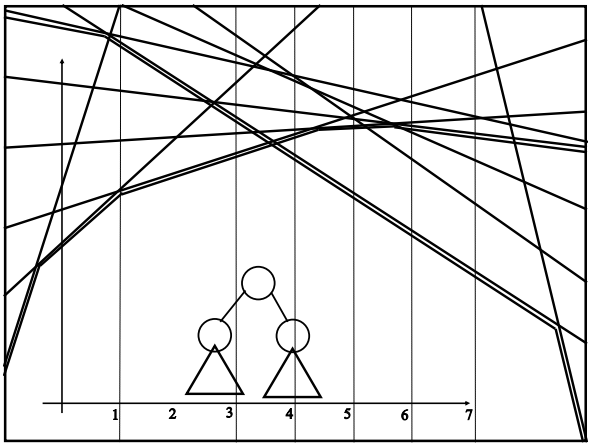


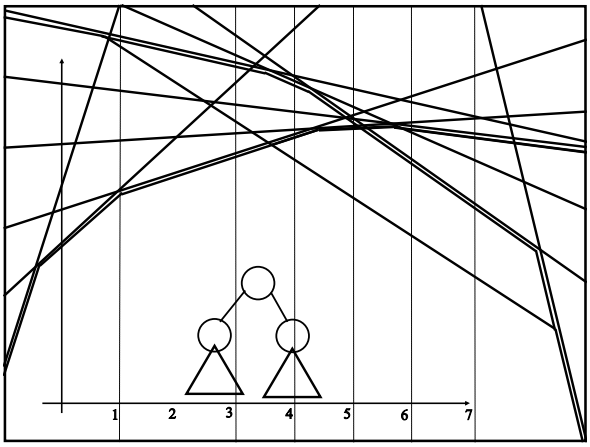




Balanced kinetic heaps

When we delete a line, lines only decrease their level.





Analysis

Key idea: Implement the envelopes using finger search trees

How much time it takes to delete an element ?

O(log n) + Σlog (# lines get on the replacement piece)
+ Σlog (# lines get off the replacement piece)

If in all nodes # lines get on/off the replacement piece < $\log^2 n$ we are ok --- the delete takes O(log n loglog n) time

However it could be that # lines get on/off the replacement piece > $\log^2 n$

Analysis (Cont)

If a line gets on the replacement piece it changes its level

Since the total number of level changes is nlog(n)

The number of chains of length > $\log^2 n$ is at most n/log(n)

So the total work in moving long chain is O(n) and can be charged to the initialization phase.

Summary so far

A better deletions only data structure

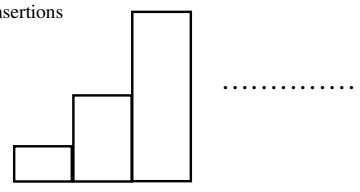
O(n) init. time and then O(log n loglog n) per delete.
O(1) find-min if we are kinetic.

Now we add insert.

Balanced kinetic heaps (finalle)

Balanced heaps (step 2):

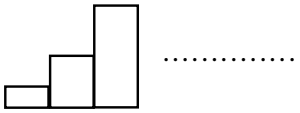
Use a well known reduction by Bently and Saxe to add insertions



Maintain a binary counter of deletion only data structures

Balanced kinetic heaps

Delete : O(log n loglog n)
Insert : O(log n)
Find-min : O(log n) (kinetic)



Open problems

Experiments: What is the best structure in practice?

Better bounds: kinetic heaps in ordinary F-heap bounds?

parametric heaps in O(log n) amortized time per operation?

Simple kinetic heaps via splay trees?